Attorney Docket No.: 1004-4282

"Express Mail" mailing label number:

EL151293368US

LEAST CHOICE FIRST ARBITER

Nils Gura
Hans Eberle

COPYRIGHT NOTICE

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates to systems having shared resources and more particularly to arbitrating multiple requests for multiple resources is such systems.

Description of the Related Art

Systems having shared resources are common. In many such systems, arbiters have to schedule usage of the shared resources to prevent conflicts resulting from requests for simultaneous access to the same shared resource. One example of such a system having shared resources susceptible to conflicts is a cross bar switch having multiple input ports and multiple output ports in which input ports make requests for connection to the output ports. Each requester or input port sends a request for an output port or a set of requests for multiple output ports to an arbiter. A particular output port may be requested by multiple input ports at the same time. Assuming an output port can be allocated to only one input port at a time, an arbitration decision is made to award the output port to one of the requesters. The arbiter chooses the requests to be granted such that resources (the output ports) are allocated to requesters in a conflict-free way. Thus, certain requests are serviced by being granted an output port and certain requests may be left unserviced in that they are denied an output port. However, the choice of which requests to grant may lead to under-utilization of the resources since some requests may be denied.

Another example of a system having shared resources is a computer system in which multiple processors are coupled to multiple memories. Assume that each processor has access to all of the memories and each memory can only be accessed by one processor at a time. When multiple processors request access to the same memory at the same time an arbitration decision has to be made as to which processor gets to access the memory in question.

While numerous arbitration schemes have been developed to try and provide fair and efficient allocation of system resources for scheduling problems that involve multiple requesters requesting multiple shared resources such as the crossbar switch or multi-processor system described above, it would be desirable to have an improved arbitration scheme that provides for high aggregate usage of the shared resources while still providing a minimum level of fairness.

SUMMARY OF THE INVENTION

Accordingly, the invention provides in one embodiment an arbiter that prioritizes requests based on the number of requests made. The highest priority is given to the requester that has made the fewest number of requests. Resources are scheduled one after the other in that a resource is allocated to the requester with the highest priority first. That is, the requester with the fewest requests (least number of choices) is chosen first. Requesters with more requests have more choices than requesters with few requests and, therefore have a reasonable likelihood of being granted one of their outstanding requests if considered after those requesters with fewer requests.

Alternatively, priority may instead be based on the number of requests made for a particular resource. In addition, priority may be based on a combination of the number of requests made by a requester and the number of requests made for a resource. Utilizing a strategy that considers fewest choices first, increases the number of granted requests and results in higher aggregate resource usage when compared with other arbitration schemes.

In one embodiment, the invention provides a method of sharing multiple resources among multiple requesters using an arbiter. The method includes receiving requests for the multiple resources from the multiple requesters. The arbiter determines respective request priorities corresponding to respective requests for respective resources made by respective requesters, each request priority is determined according to at least one of a requester priority and a resource priority, requester priority being inversely related to a number of requests made by a particular requester, resource priority being inversely related to a number of requests made for a particular resource. The arbiter allocates at least some of the resources according to requester and/or resource priorities. In order to prevent starvation, the method may also use a round robin scheme to allocate resources to requesters.

In another embodiment the invention provides an apparatus that includes a transport mechanism coupled to a plurality of resources and a plurality of requesters. An arbiter is coupled to receive a plurality of requests from the requesters, each of the requests requesting a connection between one of the requesters and at least one of the resources. The arbiter allocates resources to requesters according to at least one of a requester priority and a resource priority, the requester priority and resource priority being inversely related to, respectively, a number of requests for resources made by a respective requester and a number of requests directed to respective resources. The arbiter may further include a starvation avoidance mechanism such as a round robin mechanism to allocate resources to requesters.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention may be better understood, and its numerous objects, features, and advantages made apparent to those skilled in the art by referencing the accompanying drawings wherein use of the same reference symbols in different drawings indicates similar or identical items.

Fig. 1 shows a crossbar switch which can exploit one or more embodiments of the present invention.

Fig. 2 shows a multi-processor system which can exploit one or more embodiments of the present invention.

Figs. 3A-3C illustrates operation of one embodiment of the invention where priorities are determined according to a number of requests each requester makes.

Fig. 4 illustrates a block diagram of a hardware implementation of an arbiter according to one embodiment of the present invention.

Figs. 5A-5C illustrates operation of one embodiment of the invention where priorities are determined according to a number of requests each resource receives.

Fig. 6 illustrates a switch that utilizes a distributed embodiment of the present invention.

Fig. 7 illustrates the requests and grants passed by requesters and resources in an exemplary distributed system.

Fig. 8 illustrates exemplary vectors of requests sent from a requestor to a resource in an exemplary distributed system.

DESCRIPTION OF THE PREFERRED EMBODIMENT(S)

Referring to Fig. 1, one embodiment of the invention is illustrated in which arbiter 101 schedules usage of shared resources, i.e., output ports 103, 105 and 107 among input port requesters, 109, 111 and 113. Crossbar switch 115 forwards packets from its input ports, 109, 111 and 113 to its output ports 103, 105 and 107. Each input port can hold multiple packets destined for different output ports. The switch schedule to be calculated should connect input and output ports in such a way that as many packets as possible can be forwarded simultaneously, thus maximizing usage of shared resources. Each requester sends a set of requests to arbiter 101, which then chooses the requests to be granted such that resources are allocated to requesters in a conflict-free way.

In one embodiment of the invention, the arbiter operates in a synchronous manner in that the arbiter receives request signals 117 for shared resources at the same time from the various input ports 109, 111 and 113. Scheduling happens synchronously in that grant signals 119 are sent at the same time and the usage interval for each resource has the same length. Scheduling may be further constrained in that only one requester can use a particular resource at the same time. When developing an arbitration scheme the main goal typically is to achieve high aggregate usage of the resources while still providing a minimum level of fairness, mainly in the sense that starvation of individual requests is prevented.

In one embodiment, the arbiter makes its choices by prioritizing requesters based on the number of their requests. The highest priority is given to the requester with the fewest number of outstanding requests, and the lowest priority is given to the requester with the highest number of outstanding requests. Resources are scheduled one after the other in that a resource is allocated to the requester with the highest priority first. That way the requester with the least number of choices is chosen first. Thus, priority is inversely related to the number of requests being made. Requesters with many requests have more choices than requesters with few requests and, therefore, can be considered later and still have a reasonable likelihood of being granted one of their outstanding requests. That strategy increases the number of granted requests and, with it, results in higher aggregate usage when compared with other arbitration schemes. In one preferred embodiment, the number of outstanding requests is based only on the resources that have not yet been scheduled. That is, the number of outstanding requests is recalculated whenever a resource has been scheduled.

With reference again to Fig. 1, an example illustrates operation of one aspect of an embodiment of the least choice first arbiter. In Fig. 1, it can be seen that node 121, which is coupled to input port 109, has one request 122 for output port 103. Node 123, which is coupled to input port 111, has two requests 124, one request for output port 105 and one request for output port 107, respectively. Node 125, which is coupled to input port 113, has three requests 126, one request for each of the output ports. The arbiter receives those requests from the various input ports and prioritizes the request from node 121 (input port 109) as the highest priority request since that requester has only one request. Node 123 is prioritized as the next highest priority requester since it has two requests and node 125 is the lowest priority requester since it has three requests.

Based upon priority alone, arbiter 101 grants node 121 its request for output port 103 first. Note that while the node attached to the input port is actually making requests, for ease of description, the input ports themselves may be described herein as having requests. After arbiter 101 grants node 121 its request, priorities are recalculated. Both node 123 and node 125 have the same priority after output port 103 is scheduled, since they both have two requests (requests for output port 103 are no longer considered since that output port has been scheduled). The arbiter can now grant one of the requests of node 123 and node 125. For example, if the arbiter grants node 123 its request for output port 105, the arbiter then grants node 125 its request for output port 107. Because the arbiter serviced the requester having the least number of choices first, the number of granted requests can be maximized. To illustrate that, if another arbitration scheme had first granted node 125 its request for output port 103, a total of only two requests would have been granted. Note that the arbiter has not considered fairness in allocating the resources in the example given.

Another example where such an arbitration scheme may be utilized is shown in Fig. 2, which is a multi-bus interconnection structure 200 functioning as a transport mechanism for connecting multiple processors 201, 203 and 205 (also referred to in Fig. 2 as P0, P1 and P2) with multiple memories 207, 209 and 211 (also referred to in Fig. 2 as M0, M1 and M2). Each processor can have multiple outstanding transactions such as read and write operations. Similar to the switch embodiment described in relation to Fig. 1, the bus schedule to be calculated connects processors and memories in a conflict-free way, ideally, such that at a given time as many transactions as possible can be executed in parallel.

The arbiter 201 may be coupled to each of the buses shown in Fig. 2. The request and grant lines may be incorporated into the buses as well or may be separately coupled to the arbiter. In fact, the arbiter, rather than being separate as shown, may in fact reside in one or more of the processors or memories, or be distributed among the processors and memories in a manner described further herein.

Assuming a central arbitration scheme utilizing arbiter 201, processor P0 requests transaction T0 for memory M0 from the arbiter, processor P1 requests transactions T0 and T2 for memories M0 and M2, respectively. Processor P2 requests transactions T0, T1 and T2 to memories M0, M1 and M2, respectively. As with the switch embodiment described previously, the arbiter gives the highest priority to processor P0 since that processor has the fewest requests. After P0 is granted its T0 request, P1 has one request pending and P2 has two requests pending, since M0 has already been allocated. Thus, P1 is granted its T2 request since it has higher priority (only one request) and finally P2 is granted its T1 request. Thus, the arbiter grants P0 its T0 request, P1 its T2 request and P2 its T1 request, thus maximizing utilization of buses 200.

An additional factor to be considered in developing an arbitration scheme is to provide fairness in the sense of avoiding starvation for any particular request. Thus, a round robin scheme may be employed in addition to the arbiter prioritizing the requester with the fewest requests first. Various round robin approaches will be described further herein.

A detailed description of an arbiter is provided in the program listing in Appendix A showing procedure arbitrate, which describes one embodiment of an arbiter according to the present invention. There are two main data structures of the interfaces of procedure arbitrate. The first is the Boolean matrix R, which represents the input signals to the arbiter (line 8). R[i,j] indicates whether requester i is requesting resource j. The second main data structure is the integer array S, which represents the output values of the arbiter (line 10). S[i] indicates which resource is allocated to requester i.

The program illustrates the operation of one embodiment of the arbiter each time the arbiter receives requests from the plurality of requesters. As described previously, those requests may be received synchronously. On entry into the arbitration sequence as represented in the procedure arbitrate, the schedule S is initialized (line 16) to show no requests have been granted and the number of requests are calculated for each requester (lines 18-19). Then for each resource available to the requesters, the procedure first determines if the round robin position is asserted. If so, the request is granted. (lines 24-25).

The round robin position in the illustrated embodiment is determined by the index variables I and J along with the loop variable res. For every "schedule cycle", the requester offset I is incremented (lines 47-48). If I rolls over, the resource offset J is incremented. A schedule cycle includes the execution of the code described by the procedure arbitrate, including the round-robin determination and least choice first arbitration. Note that all resources are scheduled during a schedule cycle.

Starting with I=0 and J=0 for the first schedule cycle, the round robin positions for three requesters [0:2] and three resources [0:2] for four schedule cycles are as follows:

Schedule cycle 0: [0,0], [1,1], [2,2]
Schedule cycle 1: [1,0], [2,1], [0,2]
Schedule cycle 2: [2,0], [0,1], [1,2]
Schedule cycle 3: [0,1], [1,2], [2,0]

Thus, it can be seen that a different position in array R is designated as the start of the round robin for each schedule cycle. The "round-robin positions" within matrix R form a diagonal during every schedule cycle. The starting position for the diagonal moves downward by one element once every schedule cycle (lines 47, 48). When the index J increments, the starting position moves over to the top of the next column in array R. Note that only one of the elements of the diagonal is guaranteed to be granted (if it is asserted) per schedule cycle. That is the first round robin position in each schedule cycle.

While the arbiter has been described in the software listing of procedure arbitrate to provide a detailed example of operation of one embodiment of the arbiter, the same functionality may preferably be implemented in hardware to increase the speed with which the arbiter can respond to requests as described further herein. In other embodiments, portions of the arbiter may be implemented in hardware and other portions in software depending on the needs of a particular system. Further, the iterations of the sequential calculations of the schedule (lines 21-46), may be implemented with the help of look-ahead techniques. That is, as an alternative to the sequential approach described in Appendix A, a look-ahead technique can instead schedule resources in parallel rather than sequentially. Each resource must know what the other resources will decide. Because that information is deterministic, each resource can calculate the needed information. That parallel approach can speed up determinations.

There are additional embodiments of the arbiter. For example, the round-robin scheme can be varied in different ways. If it is desirable to guarantee that more than one element can be granted, the arbiter could consider all elements of the round robin diagonal before other requests are considered each schedule cycle. That is illustrated in detail in Appendix B.

In another embodiment, the "round-robin positions" cover a row of elements in R rather than a diagonal during each schedule cycle. That is illustrated by the following code segment, which replaces lines 24 and 25 in Appendix A.

```
If R[I,(J+res) mod MaxRes] then
  gnt := I
```

Note that with this embodiment, the requester with the requests covered by the row specified by the round robin positions is guaranteed to be granted one of its requests.

In a similar embodiment, the "round robins positions" cover a column of elements in R rather than a row during each schedule cycle. That is illustrated by the following code segment which replaces lines 24 and 25 in Appendix A.

```
If res = 0 then
begin
   r := 0;
      repeat
         if R[(r+I) mod MaxReq, J] then
            gnt := (r+I) mod MaxReq;
         r := r+1;
      until (r= MaxReq) or (gnt <> -1);
end
```

Yet another possibility is to only consider one "round-robin position" per schedule cycle as shown in the following code segment, which replaces lines 24 and 25 in Appendix A.

```
If (res = 0) and req[I,J] then
  gnt := I
```

In still another starvation avoidance approach, a round robin scheme ages requests. The longer a request waits, the higher a priority the request receives and the sooner it is considered by the round robin scheme. Any of the round-robin schemes described guarantee that every request is periodically considered. That way, starvation can be prevented. Other schemes may be implemented to prevent starvation as well. In fact, any scheme that guarantees that every request is periodically considered, prevents starvation. For example a statistical approach can be used to provide that guarantee. One example of such an approach is a random scheme that provides a probability that an element will be visited within a particular time period.

The main parameter of a round-robin scheme is the period at which elements of R are visited. For the arbiter described in Appendix A, the worst-case period is the total number of requesters times the total number of resources (MaxReq*MaxRes). The period can be shortened if the round-robin scheme can guarantee that more than one element of R (if asserted) is granted in one schedule cycle such as in the approach illustrated in Appendix B. That can be useful if the matrix R is large.

The period of the round-robin scheme also determines the minimum fraction of resource usage that a requester is guaranteed to obtain. For example, if in Figure 3A, R[2,1] is always asserted, requester 2 is guaranteed to get access to resource 1 at least once every 9th schedule cycle.

Referring again to the program listing of the procedure arbitrate in Appendix A, once the round robin position has been tested in lines 24 and 25, if the round robin position does receive a grant, then the procedure assigns the granted resource to the position in S corresponding to the requester (lines 39-45) and adjusts nrq values appropriately.

If no round robin position is true, i.e. no request is being made for that resource by that requester, then the procedure arbitrate determines for a particular resource, i.e., a particular column in R identified by [(res + J) mod MaxRes], the requester with the fewest outstanding requests and that requester is granted its requested resource (lines 29-36).

If either the round robin position or another position in array R is granted a resource, then the procedure updates array S by writing the number of the resource into the position in S corresponding to the requester (line 40). That is S[i] is written with the resource number, where i represents the requester granted the resource. The requester row in array R for the requester that was granted the resource is modified to indicate that there are no requests in that row, since only one request can be granted per requester (line 41)). The number of requests (nrq) for all the requesters requesting the granted resource is reduced because requests for a resource already allocated cannot be fulfilled and thus are not counted (lines 43-44). The number of requests (nrq) for the requester that was granted the resource is set to 0 (line 42). The procedure arbitrate then ends by incrementing I and J appropriately (lines 47-48).

The operation of the embodiment illustrated in procedure arbitrate in Appendix A will now be described with relation to Figs. 3A-3C. Assume that I and J are at [1,0] on entry into procedure arbitrate and R is as shown as shown in Fig. 3A. Array R shows that there are three requesters in the system and three resources. Requester 0 is requesting resource 0, requester 1 is requesting resources 1 and 2 and requester 3 is requesting resources 1, 2 and 3.

The number of requests is shown for each of the requesters: nrq[0] = 1, nrq[1] =2 and nrq[2]=3. The initial round robin position in R evaluated by the arbiter is R [1,0] as a result of the initial value of I and J. Because that position is not asserted (i.e. is false), procedure arbitrate checks if there are any other requests for resource 0, and determines that requester 0 and 2 are requesting resource 0. Because requester 0 has the smallest number of requests (nrq), the arbiter grants resource 0 to requester 0, as indicated by the X at R[0,0]. Procedure arbitrate zeros out row R[0,x] (made false) and reduces the number of requests for requester 2 by one (nrq[2]=2) and zeros out the number of requests by requester 0 (nrq[0]=0).

On the next iteration through the allocation loop as shown in Fig. 3B, with the loop variable res =1, the arbiter evaluates round robin position R[2,1] first. Because that position is asserted, the arbiter grants requester 2 that request as indicated by the X. The row R[2,x] is zeroed out (made false) and the number of requests for requester 1 is reduced by one (nrq[1]=1). The number of requests by requester 2 is set to 0 (nrq[2]=0). Fig. 3B illustrates that the requester 0 row was zeroed out after the requester was granted a resource in Fig. 3A. The "X" is still shown at R[0,0] to show that resource was allocated to that requester previously.

In the final allocation loop of this schedule cycle shown in Fig. 3C, with the loop variable res=2, the arbiter first evaluates round robin position R[0,2]. That position is false so the arbiter determines if there are any other requests for that resource. The only request for that resource is from requester 1 which is granted resource 2. Thus, during that schedule cycle, because the requester with the fewest choice (requester 0) was serviced first, high utilization efficiency of the resources could be achieved.

As previously stated, if speed is a consideration, which is likely in many if not most applications, a hardware implementation may be preferred. Referring to Fig. 4, a block diagram of a hardware implementation of the arbiter is illustrated. Assume there are MaxReq inputs to the arbiter. Fig. 4 illustrates hardware 400 associated with the first requester (input 0), which corresponds to requester 0 and hardware 402 associated with the last requester (MaxReq-1). The embodiment illustrated in Fig. 4 operates as follows. Each of the elements of R[i,*], which includes elements 403 and 404, is initialized, where * represents the number of resources. Thus, for example, element 403 contains all requests input by the associated requester 0 for the various resources. In addition, the array S, which includes elements 409 and 411 of the illustrated hardware, is reset. The requests for the illustrated elements 403 and 404 are summed in summers 405, 406 to generate a number of requests (nrqs) 407 and 408 for the respective requesters. If R[I+res, J+res] = 1, then grant (GNT) 410 is set to (I+res) through multiplexer 412. That is, the round robin position is granted. Otherwise, grant (GNT) 410 is set to the input with the minimum number of requests, which is determined in minimize circuit 414. If there is no request for the current output, grant is set to -1 (note that the logic to set grant to -1 is assumed to be contained within grant (GNT) 410). If grant (GNT) 410 is not -1, then the location S[gnt] (e.g. 409) is set to J+res, which indicates the resource granted to that requester. Each of the elements of R[gnt, *] is set to 0, where * represents the resources. The register 416 holding "res" is incremented so res = res + 1. The hardware continues the calculations for S until res = MaxRes, that is, the last resource is scheduled when

S[res=MaxRes-1] is evaluated. The register 418 containing the index I is incremented so I = I+1. If I = MaxReq, then I is set to 0 and the register 420 containing J is incremented. The embodiment illustrated in Fig. 4 is an illustrative block diagram and does not show all the details described. As would be known to those of skill in the art, those details along with many other hardware implementations can be provided to implement the various embodiments described herein.

In another embodiment the calculation of priorities may be varied. For example, rather than assigning priorities to the requesters as illustrated in procedure arbitrate, priorities can be assigned to the resources. More specifically, the priority of a resource is given by the number of requesters that request it. That is, the highest priority is given to the resource that is requested by the fewest number of requesters, and the lowest priority is given to the resource that is requested by the most number of requesters. Resources are assigned to the requesters one after the other.

Thus, assuming a configuration as shown in Figs. 5A-5C, with three requesters and three resources, each requester 0, 1 and 2 is evaluated in turn to see what resources are being requested and the requested resource with the highest priority is granted to that request. Assume the same round robin positions and I and J values as in Figs. 3A-3C. "Req" represents the loop variable that together with offset I, determines the requester being scheduled. In this embodiment, the arbiter is modified to determine priorities with relation to resources and not requesters.

The number of requests (nrq) for each of the resources is: nrq[0] = 1, nrq[1] =2 and nrq[2]=3. Assume the arbiter of this embodiment uses a similar round robin scheme as that used by the arbiter in Figs. 3A-3B. In Fig. 5A, the arbiter evaluates the initial round robin position R [1,0], which results from the initial value of I and J. Because that position is not asserted (i.e. is false), the arbiter according to this embodiment checks if there are any other requests by requester 1. Because requester 1 is requesting resources 1 and 2 and resource 1 has a higher priority than resource 2, the arbiter grants resource 1 to requester 1. All the requests for resource 1 are zeroed out.

On the subsequent iteration to assign resources illustrated in Fig. 5B, the arbiter evaluates the round robin position R[2,1] first. Because that position was zeroed out, the arbiter checks if requester 2 is making any other requests for resources. Requester 2 is also requesting resource 2, its only remaining request and thus the arbiter grants resource 2 to requester 2. The column representing resource 2 is zeroed out. Referring to Fig. 5C, on the final iteration which completes checking all requests in R, the arbiter evaluates round robin position R[0,2] first. Since that position is false, the arbiter then determines if any other resources are being requested by requester 0. Since requester 0 is also requesting resource 0, the arbiter grants that resource to requester 0.

In still another embodiment, the arbiter calculates priorities for both the requesters and resources, and uses those priorities to calculate a priority for each element in R. For example, the number of requests may be added together to a priority for a request, i.e., a position in R. The scheduler then iteratively grants the request with the highest priority. That way, the priority of an element in R indicates the degree of freedom of its requester as well as of the requested resource. As in other embodiments, a scheme to prevent starvation may be included. However, a scheme to prevent starvation may not always be necessary. For example, some applications may have a guaranteed pattern of requests that makes the need for starvation avoidance unnecessary.

The arbiters described so far have been for a centralized arbiter implementation. A distributed version of the least choice first arbiter is also possible. That version is distributed in that the selection of requests is performed by arbitration logic associated with the resources and requesters rather than by centralized arbiter logic. For example, assume a transport mechanism such as the simple 3X3 switch shown in Fig. 6 having three input ports and three output ports. Thus, there are three requesters and three resources. Node 601, associated with input port 0, has one request 602 for output port 0. Node 603, associated with input port 1, has two requests 604 for output ports 1 and 2, respectively. Node 605 associated with input port 2 had three requests 606, for output ports 0, 1 and 2, respectively. As shown in Fig. 6, output port 0 has requests from input port 0 and input port 2. Output port 1 has requests from input port 1 and input port 2. Output port 2 has requests from input port 1 and input port 2.

Referring to Fig. 7, showing an exemplary distributed implementation for the arbitration scheme disclosed herein, the requesters REQ 0, REQ 1 and REQ 2 supply request signals to and receive grant signals from each of the resources RES 0, RES 1 and RES 2. For example, requester REQ 0 sends request signal 701, 703 and 705 to resources RES 0, RES 1 and RES 2, respectively. Requester REQ 0 receives grant signals 707, 709 and 711 from resources RES 0, RES 1 and RES 2. In one embodiment, during an arbitration period, each resource receives request vectors indicating which requests are asserted from all requesters. A resource grants one of the received requests according to the priority scheme described herein and notifies that requester via an asserted grant signal.

The request vectors may be provided as shown in Fig. 8 where request vector 0 corresponds to input port 0, request vector 1 to input port 1 and request vector 2 to input port 2. Thus, the request vectors indicate by an asserted bit, which output port is requested. The resources can determine from the request vectors how many requests a particular requester has made. From the example shown in Figs. 6 - 8, requester REQ 0 (input port 0) has made 1 request for output port 0, requester REQ 1 (input port 1) has made 2 requests for output ports 1 and 2, and requester REQ 2 (input port 2) has made 3 requests for output ports 0, 1 and 2.

In one embodiment, a sequential scheme is utilized to implement a distributed arbiter. The sequential scheme in that embodiment requires n rounds with n being the number of resources. For each round i (i = 0..n-1):

(1) Each requester that has not yet received a grant and that is requesting resource i sends a request. The request is accompanied by a priority which is the inverse of the number of resources that the requester is requesting and that have not yet been scheduled in a previous round.

(2) The resource selects the request with the highest priority and sends a grant to the corresponding requester. If there is more than one request with the highest priority, a round-robin scheme is used to select one of the requesters. Note that all the requesters are informed by all resources when a resource has been granted so requesters know what has been scheduled in a previous round.

The rounds continue until all the resources have been scheduled.

A second embodiment uses an iterative scheme with requesters and resources making choices in parallel. In one iteration of the parallel approach according to this embodiment:

(1) Each requester that has not yet accepted a grant, sends a request to each resource that it is requesting and that has not yet been allocated. The request is accompanied by a priority which is the inverse of the number of requests the requester is sending.

(2) Each resource selects the request with the highest priority. If there is more than one request with the highest priority, a scheme such as a round robin scheme may be used to select one of the requests. A grant is sent to the requester of the selected request. The grant is accompanied by a priority which is the inverse of the number of requests the resource has received.

(3) If a requester receives more than one grant it accepts the grant with the highest priority. If there is more than one grant with the highest priority a round robin scheme is used to select the grant to be accepted. If a grant is accepted, the corresponding resource is allocated.

While in the worst case n iterations are needed to schedule all resources, a smaller number of iterations will generate schedules which are nearly optimal for most patterns of requests.

Note that either or both of the resources and requesters in a distributed approach may still implement a round robin scheme to ensure that no starvation occurs in the system. Further, the distributed arbiter may implement only the least choice priority scheme on either the resource side or the requester side.

The distributed approach described herein for the switch in Fig. 6 is equally applicable to systems requiring arbitration for resources such as the multiprocessor system illustrated in Fig. 2.

Thus, an arbiter has been described that achieves high aggregate usage of the resources while still providing a minimum level of fairness, mainly in the sense that starvation of individual requests is prevented. The arbiter makes decisions based on granting resources according to a number of requests made by a requester, number of requests made for a resource or a combination of the two. The approach described herein can be utilized in a centralized arbitration scheme or in a distributed arbitration scheme. That approach increases the number of granted requests and results in high aggregate usage of shared system resources when compared with other arbitration schemes.

The description of the invention set forth herein is illustrative, and is not intended to limit the scope of the invention as set forth in the following claims. The various embodiments may be used in various combinations with each other not particularly described. For instance, various of the starvation avoidance approaches described herein may be used in the distributed scheme. Further, the software described herein is used solely to describe operation of the system and many other data structures and hardware and software solutions can be based on the teachings herein. Those and other variations and modifications of the embodiments disclosed herein, may be made based on the description set forth herein, without departing from the scope and spirit of the invention as set forth in the following claims.

APPENDIX A

```
var
 I: 0..MaxReq-1;                              (* current round-robin requester offset *)
 J: 0..MaxRes-1;                              (* current round-robin resource offset *)
 req: 0..MaxReq-1;                            (* requester *)
 res, r: 0..MaxRes-1;                         (* resource *)
 gnt: -1..MaxReq-1;                           (* granted request *)
 min: 0..MaxRes+1;                            (* minimum number of requests *)
 R: array [0..MaxReq-1, 0..MaxRes-1] of boolean; (* R[i,j] says whether requester i is requesting resource j *)
 nrq: array [0..MaxReq-1] of 0..MaxRes;       (* number of resources requested by a requester *)
 S: array [0..MaxReq-1] of -1..MaxRes;        (* S[i] contains the granted request for requester i;
                                                   if it contains -1, no request was granted *)
procedure arbitrate;
begin
 for req := 0 to MaxReq-1 do
 begin
  S[req] := -1;                               (* initialize schedule *)
  nrq[req] := 0;
  for res := 0 to MaxRes-1 do
   if R[req,res] then nrq[req] := nrq[req]+1; (* calculate number of requests for each requester *)
 end;
 for res := 0 to MaxRes-1 do                  (* allocate resources one after the other *)
 begin
  gnt := -1;
  if R[(I+res) mod MaxReq,(J+res) mod MaxRes] then(* round-robin position wins *)
   gnt := (I+res) mod MaxReq
  else                                        (* find requester with smallest number of requests *)
  begin
   min := MaxRes+1;
   for req := 0 to MaxReq-1 do
   begin
    if (R[(req+I+res) mod MaxReq,(res+J) mod MaxRes]) and (nrq[(req+I+res) mod MaxReq] < min) then
    begin
     gnt := (req+I+res) mod MaxReq;
     min := nrq[(req+I+res) mod MaxReq];
    end;
   end;
  end;
  if gnt <> -1 then
  begin
   S[gnt] := (res+J) mod MaxRes;
   for r := 0 to MaxRes-1 do R[gnt, r] := false;
   nrq[gnt] := 0;
   for req := 0 to MaxReq-1 do
    if R[req, (res+J) mod MaxRes] then nrq[req] := nrq[req]-1;
  end;
 end;
 I := (I+1) mod MaxReq;
 if I = 0 then J := (J+1) mod MaxRes;
end;
```

APPENDIX B

```
 var
 I: 0..MaxReq-1;                                (* current round-robin requester offset *)
 J: 0..MaxRes-1;                                (* current round-robin resource offset *)
 req: 0..MaxReq-1;                              (* requester *)
 res, r: 0..MaxRes-1;                           (* resource *)
 gnt: -1..MaxReq-1;                             (* granted request *)
 min: 0..MaxRes+1;                              (* minimum number of requests *)
 R: array [0..MaxReq-1, 0..MaxRes-1] of boolean; (* R[i,j] says whether requester i is requesting resource j *)
 nrq: array [0..MaxReq-1] of 0..MaxRes;         (* number of resources requested by a requester *)
 S: array [0..MaxReq-1] of -1..MaxRes;          (* S[i] contains the granted request for requester i;
                                                        if it contains -1, no request was granted *)
procedure arbitrate;
begin
 for req := 0 to MaxReq-1 do
 begin
  S[req] := -1;                                 (* initialize schedule *)
  nrq[req] := 0;
  for res := 0 to MaxRes-1 do                   (* calculate number of requests for each requester *)
   if R[req,res] then nrq[req] := nrq[req]+1;
 end;
 for res := 0 to MaxRes-1 do                    (* check round-robin positions first *)
  if R[(I+res) mod MaxReq,(res+J) mod MaxRes] then
  begin
   gnt := (I+res) mod MaxReq;
   S[gnt] := (res+J) mod MaxRes;
   for r := 0 to MaxRes-1 do R[gnt, r] := false;
   for req := 0 to MaxReq-1 do R[req, (res+J) mod MaxRes] := false;
  end;
 for res := 0 to MaxRes-1 do                    (* allocate remaining resources 'least choice first' *)
 begin
  gnt := -1;
  min := MaxRes+1;
  for req := 0 to MaxReq-1 do                   (* find requester with smallest number of requests *)
  begin
   if (R[(req+I+res) mod MaxReq,(res+J) mod MaxRes]) and (nrq[(req+I+res) mod MaxReq] < min) then
   begin
    gnt := (req+I+res) mod MaxReq;
    min := nrq[(req+I+res) mod MaxReq];
   end;
  end;
  if gnt <> -1 then
  begin
   S[gnt] := (res+J) mod MaxRes;
   for r := 0 to MaxRes-1 do R[gnt, r] := false;
   nrq[gnt] := 0;
   for req := 0 to MaxReq-1 do
    if R[req, (res+J) mod MaxRes] then nrq[req] := nrq[req]-1;
  end;
 end;
 I := (I+1) mod MaxReq;
 if I = 0 then J := (J+1) mod MaxRes;
end;
```